W3C Candidate Recommendation

XML Pointer Language (XPointer) Version 1.0

W3C Candidate Recommendation 11 September 2001

This version:
http://www.w3.org/TR/2001/CR-xptr-20010911/ (in XML (with its own DTD, XSL stylesheet) and HTML)
Latest version:
http://www.w3.org/TR/xptr
Previous versions:
http://www.w3.org/TR/2001/WD-xptr-20010108/
Editors:
Steven DeRose, Brown University Scholarly Technology Group <Steven_DeRose@Brown.edu>
Eve Maler, Sun Microsystems <eve.maler@east.sun.com>
Ron Daniel Jr., Interwoven <rdaniel@interwoven.com>

Abstract

This specification defines the XML Pointer Language (XPointer), the language to be used as the basis for a fragment identifier for any URI reference that locates a resource whose Internet media type is one of `text/xml`, `application/xml`, `text/xml-external-parsed-entity`, or `application/xml-external-parsed-entity`.

XPointer, which is based on the XML Path Language (XPath), supports addressing into the internal structures of XML documents and external parsed entities. It allows for examination of a hierarchical document structure and choice of its internal parts based on various properties, such as element types, attribute values, character content, and relative position.

Status of this Document

This document is a Candidate Recommendation of the World Wide Web Consortium. (For background on this work, please see the XML Activity Statement.) This specification is considered stable by the XML Linking Working Group and is available for public review.

The Working Group invites implementation feedback on this specification. We expect that sufficient feedback to determine its future will have been received by 4 March 2002. Comments on this document should be sent to the public mailing list www-xml-linking-comments@w3.org (archive). While we welcome

implementation experience reports, the XML Linking Working Group will not allow early implementation to constrain its ability to make changes to this specification prior to final release.

This document is intended to be taken in conjunction with the IETF XML Media Types Internet Draft [IETF RFC 3023], which anticipates an IETF official designatation of XPointer as the fragment identifier language for resources whose type is one of one of `text/xml`, `application/xml`, `text/xml-external-parsed-entity`, or `application/xml-external-parsed-entity`. However, because of the timing problems associated with publishing two related documents on separate tracks, currently that document refers to this one only non-normatively.

It is possible that XPointer might proceed to Recommendation status, for use in sophisticated high-end pointing applications, without being incorporated into [IETF RFC 3023]. Before recommending to the IETF that the [IETF RFC 3023] Internet Draft be updated to make a normative reference to XPointer, Member feedback is urgently solicited on the following points:

1. In the course of implementing the XPointer design as presented herein, were any aspects or subparts encountered which caused significant implementation difficulty?
2. If so, would eliminating some or all of the difficult-to-implement aspects or subparts leave a coherent design still useful for your needs?
3. Does your implementation experience suggest reasons why the XPointer language as defined herein should not be recommended to the IETF as the fragment identifier language for resources of type `text/xml` etc.?

In addition to the specific points above, any feedback on patterns of implementation and use of this specification would be very welcome.

For information about the XLink language with which XPointer may be used, see http://www.w3.org/TR/xlink/.

For information about the requirements that informed development of this specification, see http://www.w3.org/TR/NOTE-xptr-req.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/. W3C publications may be updated, replaced, or obsoleted by other documents at any time. In particular it is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress".

Table of Contents

Appendices

---

1 Introduction

This specification defines the XML Pointer Language (XPointer), the language defined to express fragment identifiers for any URI reference that locates a resource whose Internet media type is one of `text/xml`, `application/xml`, `text/xml-external-parsed-entity`, or `application/xml-external-parsed-entity` [IETF RFC 3023]. This specification does not constrain the syntax or semantics of URI

references to resources of other media types, although it provides extension facilities that may be used with other types.

XPointer supports addressing into the internal structures of XML documents and external parsed entities. It allows for examination of a document's hierarchical structure and choice of its internal parts based on various properties, such as element types, attribute values, character content, and relative position. In particular, it provides for specific reference to elements, character strings, and other XML information, whether or not they bear an explicit ID attribute.

The structures located with XPointer can be used as link targets or for any other application-specific purpose. This specification does not constrain what uses an application may make of locations identified by XPointers. In particular, implementation of traversal to a resource is not constrained by this specification, and whether user "traversal" is the purpose of an XPointer at all is application-dependent. A formatted-text browser traversal might scroll to and highlight the designated location; a structure-oriented graphical viewer or a document-relationship display might do traversal in quite a different way; and a search application, parser, archival system, or expert agent might use XPointers for other purposes entirely. (The construction of linking elements in XML documents that associate arbitrary resources, including XML documents and portions thereof, is defined in a related specification, [XLink].)

XPointer is built on top of the XML Path Language [XPath], which is an expression language underlying the XSL Transformations (XSLT) language. XPointer's extensions to XPath allow it to:

- Be used in URI references to address into resources
- Address points and ranges as well as whole nodes
- Locate information by string matching

Addressing into the internal structures of DTDs and the XML declaration is outside the scope of this specification.

> **Note:**
>
> XPointer is defined to be the fragment identifier language only for resources whose media type is one of `text/xml`, `application/xml`, `text/xml-external-parsed-entity`, or `application/xml-external-parsed-entity`. However, it is expected that many applications of XML will define their own media types in order to indicate the particular usage of XML they define. (Note that [IETF RFC 3023] recommends a naming convention for specialized media types based on XML.) Schemes (discussed in **4.3 Schemes**) can be used to provide specialized addressing into the content of resources of those media types. XPointer is also suitable for addressing the generic XML aspects of those media types and as the basis for new fragment identifier languages for other XML-based media types.

1.1 Origin and Goals

In addition to XPath, the following standards have been especially influential in the development of this specification:

- *HTML* [HTML]: This system popularized an important location specifier type, the URL (now URI).
- *HyTime* [ISO/IEC 10744]: This ISO standard defines location specifier types for all kinds of data.
- *Text Encoding Initiative Guidelines* [TEI]: This application provides a formal syntax for "extended

pointers," locators for structured markup that underlie the initial design for XPointer.

The addressing components of many other hypertext systems have also informed the design of XPointer, especially [Dexter], [OHS], [FRESS], [MicroCosm], and [Intermedia].

See the XPointer Requirements Document [XPREQ] for a thorough explanation of requirements for the design of XPointer.

1.2 Notation and Document Conventions

[Definition: The key words **must**, **must not**, **required**, **shall**, **shall not**, **should**, **should not**, **recommended**, **may**, and **optional** in this specification are to be interpreted as described in [IETF RFC 2119].]

The formal grammar for XPointer is given using a simple Extended Backus-Naur Form (EBNF) notation, as described in the XML Recommendation [XML]. The circumflex (^) metacharacter used in this notation (to denote the complement of a set of characters) is not to be confused with the circumflex used to escape parenthesis characters in an XPointer.

The prototypes for XPointer functions are given using the same notation used in the [XPath] Recommendation.

This specification contains some explanatory text on the XPath language; however, such text is non-normative. In the case of conflicts, [XPath] is normative.

2 XPointer Terms and Concepts

Some special terms are defined here in order to clarify their relationship to similar terms used in the technologies on which XPointer is based. Additional XPointer-specific terms are defined in the flow of the text. Refer to [XPath], [DOM2], [XIS], and [IETF RFC 2396] for definitions of other technical terms used in this specification.

point

A position in XML information. This notion is defined fully later (see point), and comes from the DOM Level 2 [DOM2] specification's notion of positions; this specification refers to DOM positions by the term "point" to avoid confusion with XPath positions.

range

An identification of all the XML information between a pair of points. This notion is defined fully later (see range), and comes from the DOM Level 2 [DOM2] specification.

[Definition: location]

A generalization of XPath's **node** that includes points and ranges in addition to nodes.

[Definition: location-set]

An unordered list of locations, such as produced by an XPointer expression. This corresponds to the **node-set** that is produced by XPath expressions, except for the generalization to include points and ranges. Just as for a node-set, a location-set is treated as having a specific order depending on the axis that is operating on it. However, this ordering depends on XPointer's extended notion of document order as defined in **5.3.5 Document Order** , rather than XPath's original notion of document order.

[Definition: sub-resource]

The portion of an XML resource that is identified by an XPointer. For example, the whole resource being referred to is an XML document or external parsed entity, but a sub-resource might be a particular element or other location inside the document or entity. Following a link to such a sub-resource might result, for example, in highlighting or scrolling to that location in the document.

3 XPointer Processing and Conformance

This section details processing and conformance requirements on XPointers, the fragment identifiers that are based on them, and the applications that create and process them.

Should need arise to refer to the namespace for objects defined by this specification, the normative namespace URI is http://www.w3.org/2001/05/XPointer

3.1 Processing Dependencies

XPointer processing depends on [IETF RFC 2396] (as updated by [IETF RFC 2732]) processing. Fully conformant XPointer processing depends on [XPath] processing.

3.2 String Conformance

A string conforms to the XPointer specification if it adheres to the syntactic requirements and validity constraints imposed by this specification (including [XPath], whose requirements are imposed by reference). This specification does not require that the string, in association with a URI, actually point to a resource or sub-resource that exists at any given moment.

3.3 Application Conformance

[Definition: Given a resource and an XPointer, an **XPointer processor** yields the subresource identified by the XPointer, expressed as a location-set, or else an error as described in **3.4 Classes of XPointer Errors** .] This specification defines two levels of conformance of XPointer processors:

- [Definition: An XPointer processor achieves **minimal conformance** if it handles a series of XPointer parts, routing XPointer parts with particular schemes to an application that can evaluate those parts and skipping over schemes that are not understood as defined in **4.3 Schemes** ; and handles interpretation of bare-name XPointers, as prescribed in **4.2.2 Bare Names**].
- [Definition: An XPointer processor achieves **full conformance** if (in addition to the items included in minimal conformance) it handles interpretation of child-sequence XPointers and full-form XPointer parts that use the `xpointer` and `xmlns` schemes, as prescribed by this specification.]

For applications that recognize and interpret XPointers in URI references that locate resources whose Internet media type is one of `text/xml`, `application/xml`, `text/xml-external-parsed-entity`, or `application/xml-external-parsed-entity`, full conformance is required. The minimal conformance level is defined for the convenience of XML-based Internet media types [IETF RFC 3023] that choose to define fragment identifier languages based on XPointer.

3.4 Classes of XPointer Errors

The use of XPointers can give rise to several kinds of errors:

- [Definition: If the string passed to the XPointer processor does not match the syntax specified in this document, the processor yields a **syntax error**.]
- [Definition: Otherwise, if the resource passed to the processor is not a well-formed XML document entity or properly formed external parsed entity according to [XML], the processor yields a **resource error**.]
- [Definition: Otherwise, the processor attempts to evaluate the XPointer with respect to the resource, as described in **4 XPointer Model and Language** . If the evaluation fails as discussed in **4.3 Schemes** , the processor yields a **sub-resource error**.] Note that XPath allows expressions that return empty node-sets as their results and does not regard this situation as an error. Because the XPointer language is intended as a specification of locations rather than a broader query language, an empty result is an error.

This specification does not constrain how applications deal with resource errors and sub-resource errors.

4 XPointer Model and Language

XPath expressions work with a data set that is derived from the elements and other markup constructs of an XML document. The XPointer model augments this data set. Both XPointers and XPath expressions operate by selecting portions of such data sets, often by their structural relationship to other parts (for example, the parent of a node with a certain ID value). XPointers use iterative selections, each operating on what is found by the prior one.

Selection of portions of the information hierarchy is done through axes, predicates, and functions. An axis defines a sequence of candidates that might be located; predicates then test for various criteria relative to such portions; and functions generate new candidates or perform various other tasks. For example, one can select certain elements from among the siblings of some previously located element, based on whether those sibling elements have an attribute with a certain value, or are of a certain type such as "footnote"; or select the point location immediately preceding a certain "para".

4.1 Character Escaping

The XPointer language is designed to be used in the context of URI references, which require encoding and escaping of certain characters. XPointers (in URI references) also often appear in XML documents and external parsed entities, which impose some escaping requirements of their own when the encoding limits the repertoire that can be used directly. Also, because some characters are significant to XPointer processing, escaping is needed to use these characters in their ordinary sense. The following sections describe the escaping necessary for these contexts.

4.1.1 Escaping Contexts

The following contexts require various types of escaping to be applied to XPointers.

A. Escaping of XPointer-significant characters

An XPointer might need to contain characters that are significant in the XPointer syntax itself. Parentheses are significant to the XPointer language, so unbalanced parentheses must be escaped. Because the circumflex (^) is used to escape such parentheses, literal circumflexes must be escaped as described in **4.2 Forms of XPointer** .

B. Escaping of reserved URI characters

[Definition: An **internationalized URI reference**, or IURI [IURI], is a URI reference that directly uses Unicode characters [Unicode].] IURI references allow a superset of the characters of fully escaped URI references, but both must have normal occurrences of the percent sign (%) escaped because it is the character used for escaping in URIs and IURIs. Thus, when an XPointer is inserted into a URI reference or IURI reference, any occurrences of percent signs (%) are replaced with %25.

C. Converting IURI references to URI references

When an IURI reference containing an XPointer (perhaps in an XML document) is ultimately converted to a fully escaped URI reference that is suitable for resolution, any remaining characters not allowed in URI references must be escaped using the process described in **4.1.2 URI Reference Encoding and Escaping** . This process can safely be applied repeatedly to fully escaped URI references in XML documents.

Square brackets ([and]) should be replaced with %5B and %5D respectively, for compatibility with applications that have not implemented the [IETF RFC 2732] update to [IETF RFC 2396].

D. XML escaping

If an XPointer appears in a URI reference or IURI reference in an XML document or external parsed entity, any characters not expressible in the encoding used must be escaped as character references, and any characters that are significant to XML processing must be escaped as character references or as predefined entity references. This escaping is replaced when the XML document or entity is parsed.

The XPointer-processor handles undoing A. All other escaping is undone (in reverse order of application) outside the processor. If the result passed to the processor does not conform to the syntactic rules for XPointers in this specification, a syntax error results.

4.1.2 URI Reference Encoding and Escaping

The set of characters for XPointers is necessarily the same as for XML, namely [Unicode]. However, some Unicode characters are disallowed from URI references. Thus, the disallowed characters in XPointer-containing URI references must ultimately be encoded and escaped in order for URI resolver software to be able to handle them.

The disallowed characters include all non-ASCII characters, plus the excluded characters listed in Section

2.4 of [IETF RFC 2396], except for the number sign (#) and percent sign (%) and the square bracket characters re-allowed in [IETF RFC 2732]. Disallowed characters are escaped as follows:

1. Each disallowed character is converted to UTF-8 [IETF RFC 2279] as one or more bytes.
2. Any bytes corresponding to a disallowed character are escaped with the URI escaping mechanism (that is, converted to *%HH*, where HH is the hexadecimal notation of the byte value).
3. The original character is replaced by the resulting character sequence.

4.1.3 Examples of Escaping

The following table shows the progressive escaping of an XPointer containing circumflexes, double quotation marks, and spaces so that it can appear in an XML document.

Context	Notation
Initial	The desired XPointer as it was initially created: `xpointer(string-range(//P,"a little hat ^"))`
A. XPointer	With the circumflex escaped, as required by this specification: `doc.xml#xpointer(string-range(//P,"a little hat ^^"))`
B. IURI reference	Same as A (no percent sign found that needs escaping): `doc.xml#xpointer(string-range(//P,"a little hat ^^"))`
C. XML document	Double quotation marks escaped using XML's predefined `&quot;` entity reference (assuming that the XPointer appears in an attribute value delimited by double quotation marks): `doc.xml#xpointer(string-range(//P,&quot;a little hat ^^&quot;))`
D. URI reference	With occurrences of the double quotatation marks (`%22`), spaces (`%20`), and circumflexes (%5E) escaped: `doc.xml#xpointer(string-range(//P,%22a%20little%20hat%20%5E%5E%22))`

The following table shows the progressive escaping of an XPointer containing accented characters. The XPointer is intended to appear in an XML document encoded in `US-ASCII`, which does not allow the letter "é" to appear directly.

Context	Notation
Initial	The desired XPointer as it was initially created: `xpointer(id('résumé'))`
A. XPointer	Same (no circumflex or unbalanced parenthesis found that needs escaping): `xpointer(id('résumé'))`

B. IURI reference	Same as A (no percent sign found that needs escaping): `doc.xml#xpointer(id('résumé'))`
C. XML document	Represented in the `US-ASCII` encoding; accented letters are escaped with XML character references: `doc.xml#xpointer(id('r&#xE9;sum&#xE9;'))`
D. URI reference	With occurrences of the letter "é" (%C3%A9) escaped: `doc.xml#xpointer(id('r%C3%A9sum%C3%A9'))`

4.2 Forms of XPointer

This specification defines one full form of XPointer addressing and two shorthand forms. The shorthand forms are defined as abbreviations of the full form, although applications need not convert an abbreviated form into the full form before evaluation. All error conditions apply as if the full form of the XPointer were specified.

The full form is described in **4.2.1 Full XPointers** . The two short forms are bare names (Name, described in **4.2.2 Bare Names**) and child sequences (ChildSeq, described in **4.2.3 Child Sequences**). Thus, XPointer offers the following overall options.

XPointer Forms

```
[1]  XPointer  ::=  Name
                  | ChildSeq
                  | FullXPtr
```

The internal structure of an XPointer is as follows.

Internal Structure of XPointer

```
[2]   ChildSeq                ::=  Name? ('/' [1-9] [0-9]* )+
[3]   FullXPtr                ::=  XPtrPart (S? XPtrPart)*
[4]   XPtrPart                ::=  'xpointer' '(' XPtrExpr ')'
                                 | 'xmlns' '(' XPtrNsDecl? ')'
                                 | Scheme '(' SchemeSpecificExpr
                                 ')'
[5]   Scheme                  ::=  NCName
[6]   SchemeSpecificExpr      ::=  StringWithBalancedParens
[7]   StringWithBalancedParens ::=  [^()]* ('('                    [VC:
                                    StringWithBalancedParens ')'   Parenthesis
                                    [^()]*)*                       escaping ]
```

[8]	XPtrExpr	::=	Expr	[VC: Parenthesis escaping]
[9]	XPtrNsDecl	::=	NCName S? '=' S? XPtrNsURI	
[10]	XPtrNsURI	::=	Char*	[VC: Parenthesis escaping] [VC: Namespace Name]

Validity constraint: Parenthesis escaping

The end of an XPointer part is signaled by the right parenthesis ")" character that is balanced with the left parenthesis "(" character that began the part. Although the production disallows unbalanced parenthesis characters, they may occur. However, if one does occur, even within literals, it must be escaped with a circumflex (^) character preceding it. If the expression contains any literal occurrences of the circumflex, each must be escaped with an additional circumflex (that is, ^^). Any other use of a circumflex results in a syntax error.

Validity constraint: Namespace Name

The value of an XPtrNsURI must be a URI reference suitable for a namespace name.

Name, S, and Char are as defined in the XML Recommendation [XML]; NCName is as defined in the Namespaces in XML Recommendation [XML-Names]; and Expr is as defined in the XPath Recommendation [XPath], with the XPointer extensions defined in this specification.

4.2.1 Full XPointers

The full form of addressing consists of one or more [Definition: **XPointer parts**; each starts with a scheme name and is followed by a parenthesized expression, and multiple parts are optionally separated by white space.] When the scheme is `xpointer` (see **4.3 Schemes** for more information), the associated expression provides access to nodes and non-node locations in an XML document's or external parsed entity's data set.

4.2.2 Bare Names

A bare name stands for the same name provided as the argument of `id()`. For example, the first XPointer below is the bare-name equivalent of the second, which uses the full XPointer form:

```
intro
xpointer(id("intro"))
```

The bare-name shorthand is provided for two reasons:

- To encourage use of IDs, which are the form of addressing most likely to survive document change. (See [XPREQ] and [RLocs] for more information on creating robust addresses into resources.)
- To provide an analog of the HTML fragment identifier behavior for resources with XML media types. (Note that the XHTML specification [XHTML] defines some guidelines for anchors that help the bare-name shorthand to work in concert with the HTML fragment identifier in cases of content negotiation.)

4.2.3 Child Sequences

A child sequence locates an element by stepwise navigation using a sequence of integers separated by slashes (/). Each integer *n* locates the *n*th child element of the previously located element, equivalent to an XPath location step of the form `*[`*n*`]`. If the resource passed to the Xpointer-processor is a document, the first integer in the sequence will be 1, and it locates the document element; if the resource is an external parsed entity, the first integer locates one of the top-level elements.

A child sequence can optionally begin with a name before the integers; the name locates an element as described for bare names in **4.2.2 Bare Names** . For example, the following two XPointers would locate the same XML information, assuming `intro` is the ID attribute value of the element that is the fifth child of the second child of the document element:

```
intro/14/3
/1/2/5/14/3
```

4.3 Schemes

Each XPtrPart begins with a Scheme that identifies the particular notation used for that XPtrPart. This specification defines only two schemes, `xpointer` and `xmlns`. When an XPointer is the fragment identifier of an IURI and the scheme name for an XPointer part is not recognized, that XPointer part fails and processing continues with the next XPointer part (if any). The scheme mechanism provides a general framework for extensibility that can be used for future versions of XPointer, or for other media types that wish to adopt all or part of this specification in defining their own fragment identifier languages. It is recommended that if XML-based media types governed by [IETF RFC 3023] define any XPointer-style scheme names, the names beginning with "xpointer" and "xmlns" be avoided until they can be formally reserved.

For example, an XML-based vocabulary that is registered to have the `image/4Dgrafix+xml` media type might choose to adopt the XPointer scheme mechanism and define its own `4Dgrafix-xml` scheme instead of, or in addition to, the `xpointer` scheme. It is particularly useful for XML-based media types to incorporate XPointer by reference so that, in the case of content negotiation, a client that does not support the specific XML-based media type can fall back on the `xpointer` scheme if one is provided. The provision for full conformance is defined in **3.3 Application Conformance** to this end. Alternatively, XML-based media types can incorporate XPointer's notion of minimal conformance so that the scheme framework can be used, even without interpretation of the `xpointer` or `xmlns` scheme.

Minimal conformance requires the following handling. When multiple XPtrParts are provided, they must be evaluated in left-to-right order. If a part being evaluated fails for one of the following reasons, the next is evaluated:

- An unknown scheme.

- A scheme that is not applicable to the media type of the resource.
- A n XPointer part that does not locate any sub-resource present in the resource (such as an XPtrPart that returns an empty location set).

  Note that an XPointer part that uses `xmlns` scheme never returns a sub-resource and thus always fails. However, its evaluation has a potential effect on XPointer parts to its right; see **5.2.1 Namespace Initialization** for more information.

The XPointer application must consume a failed XPointer part and attempt to evaluate the next one, if any. The result of the first XPointer part whose evaluation succeeds is taken to be the sub-resource located by the XPointer as a whole. If all the parts fail, the XPointer as a whole has a sub-resource error. If a syntax error is detected in any part or in the construction of the XPointer as a whole, evaluation stops and additional parts are not consumed.

For example, the following XPointer locates the element with an ID attribute whose value is "chap1":

```
xpointer(id("chap1"))
```

However, in the absence of the DTD, it is not typically possible to determine if an attribute is of type **ID**. An XPointer like the following, with two parts, will have its first part fail if no DTD is available, but will have its second part succeed if the desired attribute's *name* is `id`; note that the intent of the second XPointer part is just an approximation of the first.

```
xpointer(id("chap1"))xpointer(//*[@id="chap1"])
```

5 XPointer Extensions to XPath

XPointer extends XPath by adding the following:

- Two new location types, `point` and `range`, corresponding to DOM positions and ranges, that can appear in location-set results; also tests (akin to node tests) for these location types.
- A generalization of the XPath concepts of nodes, node types, and node-sets to the XPointer concepts of locations (which subsume nodes, points, and ranges), and corresponding location types and location-sets.
- Rules for establishing the XPath evaluation context.
- The functions `string-range` and `range-to`, which return the range location type for selections that are not single XML nodes.
- The functions `here` and `origin`, to provide for addressing relative to the location of an XPointer expression itself, and to the point of origin for hypertext traversal when XPointers are used in that (very common) application domain.
- The functions `start-point` and `end-point`, to address the beginning and ending locations which bound another location such as a node or range.
- Allowance (as in [XSLT]) for the root node to have multiple child elements, to allow XPointers to address into arbitrary external parsed entities as well as well-formed documents.
- The functions `range` and `range-inside`, to address the covering range of locatino sets.

5.1 XPointer Additions to XPath Terms and Concepts

XPath provides for locating any subset of the *nodes* in an XML document or external parsed entities. XPath functionality, such as filtering an axis output by predicate, is generally defined in terms of operations on nodes and node-sets.

XPointer has a requirement to identify document and entity portions that are not nodes in this sense. One example of such a non-node region is an arbitrary user selection indicated by a drag between two points in a document. The two points might have different sets of ancestors in the hierarchy, or the region might form only a small part of a node. For example, a range could be a single letter or could extend from the middle of one paragraph to the middle of the next, thus containing only part of the relevant paragraphs and text nodes. Even though such locations are not nodes, XPointer needs to be able to apply XPath operations to them as well as to nodes.

To accomplish this, XPointer defines **location** as a generalization of XPath's **node**. Every location is either a point, a range, or an XPath node. Thus, XPointer also defines **location-set** as a generalization of XPath's node-set. All locations generated by XPath constructs are nodes; XPointer constructs can also generate points and ranges.

> **Note:**
>
> The order of characters displayed on a computer screen might not reflect their order in the underlying XML document, for example, when a portion of a right-to-left language such as Arabic is embedded in a left-to-right language such as French. For XPointers that identify ranges of strings, the document order is used, not the display order. Thus, an XPointer for a single range might be displayed non-contiguously, and conversely a user selection of an apparent single range might correspond to multiple non-contiguous XPointer ranges in the underlying document.

5.2 Evaluation Context Initialization

An XPointer is evaluated to yield an object of type location-set. This evaluation is carried out within a context identical to the XPath evaluation context, except for the generalization of nodes to locations. XPointer applications must initialize the evaluation context as described in this section before evaluating an XPtrExpr.

An XPointer evaluation context contains the following information:

- A location (the **context location**), initialized to the root node of an XML document or external parsed entity. When the XPointer is a fragment identifier of a URI reference, the document or external parsed entity is the one identified by the URI portion. If an application uses XPointers in another context than as a URI reference's fragment identifier, this specification does not constrain how the applicable document or external parsed entity is chosen.
- A non-zero context position, initialized to 1.
- A non-zero context size, initialized to 1. (At the start, the only location in the current location list is the context location.)
- A set of variable bindings. No means for initializing these is defined for XPointer applications. Thus, the set of variable bindings used when evaluating an XPointer is empty, and use of a variable reference in an XPointer results in a syntax error.
- A library of functions. Only functions defined in XPath or XPointer can be used in XPointers. An XPointer that uses other functions results in a syntax error.
- A set of namespace declarations, described in **5.2.1 Namespace Initialization** .

- When applicable, properties for the locations that the `origin` and `here` functions return.

5.2.1 Namespace Initialization

For any XPointer part that uses the `xpointer` scheme, the evaluation context of that part must be initialized to a set of namespace declarations consisting of a declaration of the `xml` prefix, bound to the URI `http:/www.w3.org/XML/1998/namespace`, plus any namespace declarations specified by `xmlns` XPointer parts appearing to its left. Each `xmlns` part defines a namespace declaration as a prefix (NCName) and namespace URI (XPtrNsURI). In the event that two or more `xmlns` parts specify the same prefix, the rightmost one is used. Any `xmlns` parts attempting to override the `xml` prefix must be ignored.

As an example, assume the following target XML resource:

```
<doc>
   <x:a xmlns:x="http://example.com/foo">
     <x:a xmlns:x="http://example.org/bar">This element and
     its parent are in different namespaces.</x:a>
   </x:a>
</doc>
```

Evaluation of the following XPointer will fail; it cannot be known which a element is desired because no namespace declarations are in scope:

```
xpointer(//x:a)
```

The following avoids this evaluation failure and ensures that the outer a element is addressed:

```
xmlns(x=http://example.com/foo) xpointer(//x:a)
```

The following (printed on two lines for convenience) allows for accurate addressing of the inner a element; note that the prefix used inside the XPointer need not correspond to the prefix (if any) actually used in the target resource:

```
xmlns(x=http://example.com/foo) xmlns(y=http://example.org/bar)
xpointer(//x:a/y:a)
```

5.3 The `point` and `range` Location Types

To address non-node locations, XPointer defines two new location types, `point` and `range`, that can appear in location-sets and can be operated on by XPath node tests and predicates. Locations of the `point` and `range` type represent positions and ranges as in DOM Level 2 [DOM2]. This section defines the `point` and `range` types and their characteristics required for XPath interoperability.

Note:

Unlike DOM Level 2, which is based on UTF-16 units, XPath and XPointer are based on UCS

characters. So while the concepts of points and ranges are based on the DOM 2 notions of positions and ranges, there are differences in detail. For example, a sequence which in DOM counts as two characters might count in XPointer as one character.

Points and ranges can be used as XPointer context locations. This allows the XPointer `[]` operator to be used to select from sets of ranges. Also, a point as a context location, when followed by a `range-to` function, selects a range.

5.3.1 Definition of Point Location

[Definition: A location of type **point** is defined by] [Definition: a node, called the **container node**, and a non-negative integer, called the **index**.] It can represent the location preceding or following any individual character, or preceding or following any node in the data set constructed from an XML document or external parsed entity. Two points are identical if they have the same container node and index.

Note:

This specification does not constrain the implementation of points; applications need not actually represent points using data structures consisting of a node and an index.

Also note that, while some nodes containing points have explicit boundaries (such as element start-tags and end-tags), the boundaries of text nodes are implicit. Applications that present a graphical user interface for the selection or rendering of points and ranges need to take into consideration the fact that some seemingly identical points, such as the points just inside and just outside the closing boundary of a text node inside an element, are in fact distinguished.

[Definition: When the container node of a point is of a node type that can have child nodes (that is, when the container node is an element node or a root node), then the index is an index into the child nodes; such a point is called a **node-point**.] The index of a node-point must be greater than or equal to zero and less than or equal to the number of child nodes of the container. An index of zero indicates the point before any child nodes, and a non-zero index *n* indicates the point immediately after the *n*th child node.

Note:

The zero-based counting of node-points differs from the one-based counting of `string-range` and other XPointer and XPath functions.

[Definition: When the container node of a point is of a node type that cannot have child nodes (i.e., text nodes, comments, processing instructions, attribute nodes, and namespace nodes), then the index is an index into the characters of the string-value of the node; such a point is called a **character-point**.] The index of a character-point must be greater than or equal to zero and less than or equal to the length of the string-value of the node. An index of zero indicates a point immediately before the first character of the string-value, and a non-zero index *n* indicates the point immediately after the *n*th character of the string-value.

A point location does not have an expanded-name.

The string-value of a point location is empty.

The axes of a point location are defined as follows:

- The **child**, **descendant**, **preceding-sibling**, **following-sibling**, **preceding**, **following**, **attribute**, and **namespace** axes are empty.
- The **descendant-or-self** axis contains the point itself.
- The **self** axis contains the point itself.
- The **parent** axis contains the point's container node.
- The **ancestor** axis contains the point's container node and its ancestors.
- The **ancestor-or-self** axis contains the point itself, the point's container node, and its ancestors.

5.3.2 Definition of Range Location

A location of type [Definition: range] is defined by two points, a start point and an end point. A range represents all of the XML structure and content between the start point and end point. This is distinct from any list of nodes and/or characters, in part because some nodes might be only partly included. The start point and end point of a range must be in the same document or external parsed entity. The start point must not appear after the end point in document order (see **5.3.5 Document Order**).

[Definition: A range whose start point and end point are equal is a **collapsed range.**]

If the container node of one point of a range is a node of a type other than element, text, or root, the container node of the other point of the range must be the same node. For example, it is allowed to specify a range from the start of a processing instruction to the end of an element, but not to specify a range from text inside a processing instruction to text outside it.

A range location does not have an expanded-name.

The string-value of a range location consists of the characters that are in text nodes and that are between the start point and end point of the range.

The axes of a range location are identical to the axes of its start point. For example, the **parent** axis of a range contains the parent of the start point of the range.

> **Note:**
>
> The `start-point` and `end-point` functions can be used to navigate with respect to the boundaries of a range location.

5.3.3 Covering Ranges for All Location Types

[Definition: A **covering range** is a range that wholly encompasses a location. For every location, a covering range is defined as follows:]

- For a range location, the covering range is identical to the range.
- For a point location, the start and end points of the covering range are the point itself.
- For an attribute or namespace location, the container node of the start point and end point of the covering range is the attribute or namespace location; the index of the start point of the covering range is 0; and the index of the end point of the covering range is the length of the string-value of the attribute or namespace location.
- For the root location, the container node of the start point and end point of the covering range is the root node; the index of the start point of the covering range is 0; and the index of the end point of the

covering range is the number of children of the root location.
- For any other kind of location, the container node of the start point and end point of the covering range is the parent of the location; the index of the start point of the covering range is the number of preceding sibling nodes of the location; and the index of the end point is one greater than the index of the start point.

5.3.4 Tests for `point` and `range` Locations

XPointer extends the XPath production for NodeType... by adding items for the `point` and `range` location types. The production (number 38 in XPath) becomes as follows:

NodeType

```
[11]  NodeType  ::=  'comment'
                   | 'text'
                   | 'processing-instruction'
                   | 'node'
                   | 'point'
                   | 'range'
```

This definition allows NodeTests to select locations of type `point` and `range` from a location-set that might include locations of many types.

5.3.5 Document Order

XPointer extends XPath's concept of **document order** to cover point and range locations. The concept applies equally in external parsed entities.

A point can be either a node point or a character point. Conceptually, node points label gaps between nodes, while character points occur within a node, between the node points to the right and left of the node.

Thus, an element P has a node point before and after it. If the P element contains sub-elements and/or text nodes, there are node points for the gap before the first child node, between each successive pair of child nodes, and after the last child node; they are numbered in order from 0.

WIthin any text node, there are character points preceding the first character of the text, between each successive pair of characters, as well as after the last character; they are numbered in order from 0.

For any point, there is an [Definition: immediately preceding node] defined as follows (except that there is no point defined preceding or following the root):

- For a node-point with a non-zero index *n*, the immediately preceding node is the *n*th child of the node-point's container node.
- For a node-point with a zero index, the immediately preceding node is the container node unless it has any attribute or namespace nodes. If the container node does have attribute or namespace nodes, then the immediately preceding node is the last of those attribute or namespace nodes (note that the order of attribute and namespace nodes is implementation-dependent).

- For a character-point, the immediately preceding node is the container node of the character-point.

The following diagram illustrates the relation between container nodes, node-points and character-points.

Tree-structured diagram of XML document fragment, illustrating character and node points

The document order of locations is specified here according to the location types to be compared:

Node and node

As defined by XPath.

Node and point

A node and point can never be equal in document order. A node is before a point if the node is before or equal in document order to the immediately preceding node of the point; otherwise, the node is after the point.

Node and range

A node and range can never be equal in document order. A node is before a range if the node is before the start point of the range; otherwise the node is after the range.

Point and point

Two points P1 and P2 are equal if their immediately preceding nodes are equal and the indexes of the points are equal. P1 is before P2 if P1's immediately preceding node is before P2's, or if their immediately preceding nodes are equal and P1's index is less than P2's. Otherwise P2 is after P1.

Point and range

A point P is equal to a range R if R's start and end points are both equal to P; otherwise P is before R if P is before or equal to the start point of R; otherwise P is after R.

Range and range

Two ranges R1 and R2 are equal in document order if their start points are equal and their end points are equal. R1 is before R2 if R1's start point is before R2's start point or if R1's start point is equal to R2's and R1's end point is before R2's; otherwise R2 is after R1.

Note that one consequence of these rules is that a point can be treated the same as the equivalent collapsed range.

5.4 XPointer Functions

XPointer adds the following functions to those in XPath; these functions must be provided by XPointer applications.

5.4.1 `range-to` Function

location-set **range-to**(*location-set*)

For each location in the context, `range-to` returns a range. The start point of the range is the start point of the context location (as determined by the `start-point` function), and the end point of the range is the end point (as determined by the `end-point` function) of the location found by evaluating the expression argument with respect to that context location.

The change made to the XPath syntax to support the `range-to` construct corresponds to a single addition to the Step production of the [XPath] specification. The original production is as follows:

```
[4] Step ::= AxisSpecifier NodeTest Predicate*
               | AbbreviatedStep
```

The XPointer version is as follows:

```
[4xptr] Step ::= AxisSpecifier NodeTest Predicate*
                   | AbbreviatedStep
                   | 'range-to' '(' Expr ')' Predicate*
```

This change is a single exception for the `range-to` function. It is not a generic change and is not extensible to other functions. The modified production expresses that a range computation must be made for each of the locations in the current location list.

As an example of using the `range-to` function, the following XPointer locates the range from the start point of the element with ID "chap1" to the end point of the element with ID "chap2".

```
xpointer(id("chap1")/range-to(id("chap2")))
```

As another example, imagine a document that uses empty elements (such as <REVST/> for revision start and <REVEND/> for revision end) to mark the boundaries of edits. The following XPointer would select, for each revision, a range starting at the beginning of the REVST element and ending at the end of the next REVEND element:

```
xpointer(descendant::REVST/range-to(following::REVEND[1]))
```

5.4.2 `string-range()` Function

location-set **string-range**(*location-set*, *string*, *number*?, *number*?)

For each location in the *location-set* argument, `string-range` returns a set of ranges determined by searching the string-value of the location for substrings that match the *string* argument. An empty string is defined to match before each character of the string-value and after the final character.White space in a string is matched literally, with no normalization except that provided by XML for line ends and attribute values. Each non-overlapping match can contribute a range to the resulting location set.

The third argument gives the position of the first character to be in the resulting range, relative to the start of the match. The default value is 1, which makes the range start immediately before the first character of the matched string. The fourth argument gives the number of characters in the range; the default is that

the range extends to the end of the matched string. Thus, both the start point and end point of each range returned by the `string-range` function will be character points.

Element boundaries, as well as entire embedded nodes such as processing instructions and comments, are ignored as specified by the definition of string-value in [XPath].

For any particular match, if the *string* argument is not found in the string-value of the location, or if the third and fourth argument indicates a range that is wholly beyond the beginning or end of the document or entity, then no range is added to the result for that match.

The start and end points of the range-locations in the returned location-set will all be character points.

For example, the following expression returns a range that selects the 17th occurrence of the string "Thomas Pynchon" occurring in a `title` element:

```
string-range(//title,"Thomas Pynchon")[17]
```

As another example, the following expression returns a collapsed range whose points immediately precede the letter "P" (8 from the start of the string) in the third occurrence of the string "Thomas Pynchon" in a `P` element:

```
string-range(//P,"Thomas Pynchon",8,0)[3]
```

Alternatively this could be specified as follows:

```
string-range(string-range(//P,"Thomas Pynchon")[3],"P",1,0)
```

String-values are "views" into only the string content of a document or entity; they do not retain the structural context of any non-text nodes interspersed with the text. Because the `string-range` function operates on a string-value, markup that intervenes in the middle of a string does not prevent a match. (Note that for this reason, a `string-range` match is a range describing the relevant substring of the string-value, not necessarily a contiguous string in a single text node in the document.) For example, if the 17th occurrence of "Thomas Pynchon" had some inline markup in it as follows, it would not change the string returned by the XPointer application:

```
Thomas <em>Pyn</em>chon
```

The following expression selects the fifth exclamation mark in any text node in the document and the character immediately following it:

```
string-range(/,"!",1,2)[5]
```

Although these examples locate ranges via text in the string-values of elements, `string-range` is useful for locating ranges that are wholly enclosed in other node types as well, such as attributes, processing instructions, and comments.

5.4.3 Additional Range-Related Functions

The following functions are related to ranges.

5.4.3.1 `range` *Function*

location-set **range**(*location-set*)

The `range` function returns ranges covering the locations in the argument location-set. For each location *x* in the argument location-set, a range location representing the covering range of *x* is added to the result location-set.

5.4.3.2 `range-inside` *Function*

location-set **range-inside**(*location-set*)

The `range-inside` function returns locations covering the contents of the locations in the argument location-set. For each location *x* in the argument location-set, a location is added to the result location-set. If *x* is a range location or a point, then *x* is added to the result location-set. Otherwise *x* is used as the container node of the start and end points of the range location to be added, which is defined in this way: The index of the start point of the range is zero. If the end point is a character point then its index is the length of the string-value of *x*; otherwise its index is the number of children of *x*.

5.4.3.3 `start-point` *Function*

location-set **start-point**(*location-set*)

For each location *x* in the argument location-set, `start-point` adds a location of type point to the resulting location-set. That point represents the start point of location *x* and is determined by the following rules:

- If *x* is of type point, the resulting point is *x*.
- If *x* is of type range, the resulting point is the start point of *x*.
- If *x* is of type root, element, text, comment, or processing instruction, the container node of the resulting point is *x* and the index is 0.
- If *x* is of type attribute or namespace, the XPointer part in which the function appears fails.

5.4.3.4 `end-point` *Function*

location-set **end-point**(*location-set*)

For each location *x* in the argument location-set, `end-point` adds a location of type point to the result location-set. That point represents the end point of location *x* and is determined by the following rules:

- If *x* is of type point, the resulting point is *x*.
- If *x* is of type range, the resulting point is the end point of *x*.
- If *x* is of type root or element, the container node of the resulting point is *x* and the index is the number of children of *x*.
- If *x* is of type text, comment, or processing instruction, the container node of the resulting point is *x*

and the index is the length of the string-value of x.
- If *x* is of type attribute or namespace, the XPointer part in which the function appears fails.

5.4.4 `here` Function

location-set **here**()

The `here` function is meaningful only when the XPointer being interpreted occurs in an XML document or external parsed entity; otherwise the XPointer part in which the `here` function appears fails. When in an XML context, the `here` function returns a location-set with a single member. There are two possibilities for the location returned:

- If the XPointer being evaluated appears in a text node inside an element node, the location returned is the element node.
- Otherwise, the location returned is the node that directly contains the XPointer being evaluated.

In the following example, the `here` function appears inside an XPointer that is in an attribute node. The XPointer as a whole, then, returns the `slide` element just preceding the `slide` element that most directly contains the attribute node in question.

```
<button
   xlink:type="simple"
   xlink:href="#xpointer(here()/ancestor::slide[1]/preceding::slide[1])">
Previous
</button>
```

Note:

The type of the node in which the `here` function appears is likely to be `text`, `attribute`, or `processing-instruction`. The returned location for an XPointer appearing in element content does not have a node type of `element` because the XPointer is in a text node that is itself inside an element.

5.4.5 `origin` Function

location-set **origin**()

The `origin()` function is meaningful only when the XPointer is being processed in response to traversal of a link expressed in an XML document. The `origin` function enables addressing relative to third-party and inbound links such as defined in XLink. This allows XPointers to express relative locations when links do not reside directly at one of their endpoints. The function returns a location-set with a single member, which locates the element from which a user or program initiated traversal of the link. (See [XLink] for information about traversal.)

It is a resource error to use `origin` in the fragment identifier portion of a URI reference where a URI is also provided and identifies a resource different from the resource from which traversal was initiated, or in a situation where traversal is not occurring.

5.5 Root Node Children

[XML] requires well-formed documents to contain a single element at the top level. Thus, the XPath data model of a well-formed document will have a root node with a single child node of type element. In order to allow XPointer to be used to address locations in arbitrary external parsed entities, along with well-formed documents, XPointer extends the XPath data model to allow the root node to have any sequence of nodes as children that would be possible of an element node. This extension is identical to the one made by [XSLT]. Thus, the root node may contain child nodes of type text, and any number of child nodes of type element.

A References

A.1 Normative References

IETF RFC 2119
RFC 2119: Key words for use in RFCs to Indicate Requirement Levels. Internet Engineering Task Force, 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

IETF RFC 2279
RFC 2279: UTF-8, a transformation format of ISO 10646. Internet Engineering Task Force, 1998. (See http://www.ietf.org/rfc/rfc2279.txt.)

IETF RFC 2396
RFC 2396: Uniform Resource Identifiers. Internet Engineering Task Force, 1995. (See http://www.ietf.org/rfc/rfc2396.txt.)

IETF RFC 2732
RFC 2732: Format for Literal IPv6 Addresses in URL's. Internet Engineering Task Force, 1999. (See http://www.ietf.org/rfc/rfc2732.txt.)

IETF RFC 3023
RFC 3023: XML Media Types. Internet Engineering Task Force, 2001. (See http://www.ietf.org/rfc/rfc3023.)

DOM2
Document Object Model (DOM) Level 2 Specification: Version 1.0. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/DOM-Level-2-Core/.)

Unicode
The Unicode Consortium. *The Unicode Standard.* (See http://www.unicode.org/unicode/standard/standard.html.)

XML
Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, editors. *Extensible Markup Language (XML) 1.0 (Second Edition).* World Wide Web Consortium, 2000. (See http://www.w3.org/TR/REC-xml.)

XML-Names
Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/REC-xml-names/.)

XPath
James Clark and Steve DeRose, editors. *XML Path Language (XPath)*. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/xpath.)

A.2 Non-Normative References

CHUM
Steven J. DeRose and David G. Durand. 1995. "The TEI Hypertext Guidelines." In *Computing and*

the Humanities 29(3). Reprinted in *Text Encoding Initiative: Background and Context*, ed. Nancy Ide and Jean Veronis, ISBN 0-7923-3704-2.

CMOD

Martin Dürst and François Yergeau, editors. *Character Model for the World Wide Web*. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/charmod/.)

Dexter

Halasz, Frank. 1994. "The Dexter Hypertext Reference Model." In *Communications of the Association for Computing Machinery* 37 (2), February 1994: 30-39.

ERR

XPointer Errata. This document will be created when XPointer goes to Recommendation. Until that time, check the public XML Linking page at http://www.w3.org/XML/Linking.html. (See .)

FRESS

Steven J. DeRose and Andries van Dam. 1999. "Document structure in the FRESS Hypertext System." *Markup Languages 1* (1) Winter. Cambridge: MIT Press: 7-32. (See http://www.stg.brown.edu/~sjd/fress.html.)

HTML

HTML 4.01 Specification. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/html4/.)

Intermedia

Yankelovich, Nicole, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. 1988. "Intermedia: The Concept and the Construction of a Seamless Information Environment." *IEEE Computer* 21 (January, 1988): 81-96.

ISO/IEC 10744

ISO/IEC 10744-1992 (E). Information technology --Hypermedia/Time-based Structuring Language (HyTime). Geneva: International Organization for Standardization, 1992. *Extended Facilities Annex.* [Geneva]: International Organization for Standardization, 1996. (See http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html.)

IURI

Internationalized URIs. Expired Internet-Draft. Internet Engineering Task Force, 2000. (See http://www.w3.org/International/2000/03/draft-masinter-url-i18n-05.txt.)

MicroCosm

Hall, Wendy, Hugh Davis, and Gerard Hutchings. 1996. *Rethinking Hypermedia: The Microcosm Approach.* Boston: Kluwer Academic Publishers. ISBN 0-7923-9679-0.

OHS

van Ossenbruggen, Jacco, Anton Eliëns and Lloyd Rutledge. "The Role of XML in Open Hypermedia Systems." Position paper for the 4th Workshop on Open Hypermedia Systems, ACM Hypertext '98. (See http://aue.auc.dk/~kock/OHS-HT98/Papers/ossenbruggen.html.)

RLocs

Thomas A Phelps and Robert Wilensky. *Robust Intra-document Locations.* University of California, Berkeley. (See http://www.cs.berkeley.edu/~phelps/Robust/papers/RobustHyperlinks.html.)

TEI

C. M. Sperberg-McQueen and Lou Burnard, editors. *Guidelines for Electronic Text Encoding and Interchange*. Association for Computers and the Humanities (ACH), Association for Computational Linguistics (ACL), and Association for Literary and Linguistic Computing (ALLC). Chicago, Oxford: Text Encoding Initiative, 1994. (See http://www.uic.edu/orgs/tei/p3/.)

XHTML

XHTML 1.0: The Extensible HyperText Markup Language. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xhtml1/.)

XIS

John Cowan and Richard Tobin, editors. *XML Information Set*. World Wide Web Consortium, 2001. (See http://www.w3.org/TR/xml-infoset/.)

XLink

Steve DeRose, Eve Maler, David Orchard, and Ben Trafford, editors. *XML Linking Language (XLink)*. World Wide Web Consortium, 2000. (See http://www.w3.org/TR/xlink/.)

XPREQ

Steve DeRose, editor. *XML XPointer Language Requirements Version 1.0*. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/NOTE-xptr-req.)

XSLT

James Clark, editor. *XSL Transformations (XSLT) Version 1.0*. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/xslt.)

B Working Group Members (Non-Normative)

The first working drafts of this specification were developed in the XML Working Group, whose members are listed in [XML]. The work was completed in the XML Linking Working Group, with the following members active at the completion of this specification:

- Peter Chen, LSU, Bootstrap Alliance
- Ron Daniel, Interwoven (*XPointer co-editor*)
- Steven DeRose, invited expert (*XPointer co-editor*)
- David Durand, University of Southhampton, Dynamic Diagrams
- Masatomo Goto, Fujitsu Laboratories
- Paul Grosso, Arbortext
- Chris Maden, Lexica
- Eve Maler, Sun Microsystems (*past co-chair and co-editor*)
- Jonathan Marsh, Microsoft
- David Orchard, Jamcracker
- Henry Thompson, W3C and University of Edinburgh (*co-chair and W3C staff contact*)
- Daniel Veillard, W3C staff contact (*co-chair*)

The editors wish to acknowledge substantial contributions from Tim Bray, who previously served as co-editor and co-chair. We would also like to acknowledge substantial contributions from James Clark, especially on the integration with XPath. We would like to thank Gavin Nicol and Martin Dürst for help with passages related to internationalization. Finally, we would like to thank the XML Linking Interest Group and Working Group for their support and input.